**Finite Automata**

Source Code //x = 3.157\*(1.43 + y);

| //(Subset of) ASCII

| //"Characters" from an "Alphabet"

| //REGULAR LANGUAGE < Finite Automaton!

| //A regular language is any language that can be identified by a finite automaton

|

V

LEXER //LEXical analyzER

| //is a Finite State Machine that can “recognize” “sentences” in a regular language

| //sentence: entire string of characters (your source code)

| //recognize: find errors in sentence that isn’t in the language (ex. -3.14E+31 VS. -31.4E\*31)

| //if error found, stops. Makes suggest as to what is wrong and where.

| Lexeme: if you make it any smaller, it loses its meaning (tokens)

| x = 3.157\*(1.43 + y);

V z += x \* -y

|  |  |
| --- | --- |
|  | Attribute (value) |
| id | x |
| assign |  |
| FNUM | 3.157 |
| mult |  |
| oparen |  |
| FNUM | 1.43 |
| ADD |  |
| id | Y |
| cparen |  |
| semi |  |

Token Stream //Context Free Grammar < Push Down Automaton

| //Push Down Automaton is a finite automaton

| married with a stack

|

|

|

|

|

|

|

|

|

|

V

Parser //Syntax Analyzer

|

|

V

Parse Tree

|

|

V

Code Generator

|

|

V

Executable //Intermediate Language, ex. In java, this is bite code then used in java run time environment

**EXAMPLE:**

Our Alphabet:

a b c

Rules for building sentence:

aab aac aac aab 🡪 (aab|aac)+

aab -> possible phrase

aac -> possible phrase

| -> or

+ -> atleast one, but may have more

\* -> zero or more

Start state (S) ---a---> State (S 1) ---a---> State (S2 ) ---b---> State ((S3)) ---a--->(S1)

\ ---c---> State ((S4)) ---a--->(S1)

OR

Start state (S) ---a---> State (S 1) ---a---> State (S2 ) ---b, c---> State ((S3)) ---a--->(S1)

(( )) – Part of language if string ends here.

All states lead to S% (---\*--->(S5)) meaning end that isn’t a valid string (no where to go)

not written on graph though due to clutter.

Now with \*

(aab|aac)\*

Start state ((S)) ---a---> State (S 1) ---a---> State (S2 ) ---b, c---> State ((S))

Comments in C

//Comment or /\*Comment \*/

Syntax error: /\* … /\*… \*/ … \*/

Lexer Diagram (Example 2):

Note: “-\*-“ equals ANYTHING ELSE

/ -\*-\

Start state ((S)) ---‘/’---> State (S 1) ---‘/’---> State ((S2) )---‘\n’--->((S))

\ -\*-/ \ \ ---‘\*’---> State (S3) ---‘\*’--->(S4)

\ -\*-> ((S) \ -\*-/ \ \ ---‘/’---> ((S))

\-\*->(S3)

\*\*\*THIS IS OUR NEXT C HOMEWORK

Recognize INT & FLT

y = z + 9;

x = 3.14 + y;

\*Pull out, and identify these.